IN THE CLAIMS

This is a complete and current listing of the claims, marked with status identifiers in parentheses. The following listing of claims will replace all prior versions and listings of claims in the application.

1. (Currently Amended) A method for modifying software, comprising:

~~wherein~~ initially forming, ~~on the vendor side,~~ from an original piece of software ~~consisting~~ including only ~~of~~ source text ~~(SC1, SC2, SC)~~, a hybrid form of ~~said~~ the original software, ~~is~~ formed in such a way that at least one part ~~(SC1)~~ of the source text is compiled into at least one of a byte ~~or~~ and binary code ~~(B1,..,B)~~ and at least one further part ~~(SC2)~~ of the source text is converted into a code ~~(CodeML)~~ formulated in a meta markup language for at least one variation point ~~(VP2,..VP),~~;

~~wherein~~ subsequently converting~~, on the customer side,~~ only at least one variation point ~~(VP2)~~ of the hybrid form ~~(SW)~~ of the original software ~~is converted~~ as necessary by ~~means of~~ a transformation ~~(T)~~ in accordance with transformation rules ~~(TR)~~ into at least one other code ~~(CodeML*)~~ formulated in the meta markup language; and

~~wherein~~ forming ~~said other code (CodeML*) directly forms~~ a modified variation point ~~(VP2*)~~ of an adapted piece of at least one of software and ~~(SW*) or~~ a source code ~~(SC*) is formed~~ from said other code ~~(CodeML*) by means of~~ via a converter ~~(RCONV)~~ and then forming at least noe of a binary ~~or~~ and byte code ~~(B*)~~ of the modified variation point ~~(VPB2)~~ of an adapted piece of software ~~(SW*) is formed by means of~~ via a compiler ~~(COMP)~~, ~~with~~ the original ~~(SW)~~ and the adapted software ~~(SW*)~~ differing in terms of at least one of their program execution and ~~/or~~ program content.

2. (Currently Amended) The method as claimed in claim 1, wherein the transformation rules ~~(TR)~~ have at least one modification rule for a variation point.

3. (Currently Amended) The method as claimed in claim 1 ~~or 2~~, wherein the modification rule initiates an update to ~~a~~ at least one of a more recent software version or a patching operation.

4. (Currently Amended) The method as claimed in claim 1 ~~or 2~~, wherein the modification of at least one variation point ~~(VP)~~ is performed by ~~means~~ way of the transformation at runtime.

5. (Currently Amended) The method as claimed in ~~one of the preceding claims~~claim 1,
wherein the programming language of the source code is Java and the meta markup language of the variation points is XML and wherein the transformation and the rule description are implemented ~~by means of~~via XSLT and XSL.

6. (Currently Amended) A method for modifying source code, comprising:
~~wherein~~ making a first code ~~(CodeML)~~ formulated in a meta markup language with language extensions ~~(LE)~~ formulated in at least one meta markup language ~~is~~ available as the source code~~,~~;
~~wherein~~ converting the source code, ~~is converted by means of~~via a transformation ~~(T)~~ in accordance with transformation rules, ~~(TR)~~ into a second code ~~(CodeML*)~~ formulated in the meta markup language without the language extensions ~~(LE)~~ formulated in the meta markup language~~,~~;
~~wherein~~ using the transformation rules to form a language converter ~~(LC)~~ which at least one of resolves and~~or~~ applies

the language extensions ~~(LE)~~ of the first code in such a way that they can be processed by a back-converter ~~(RCONV)~~ that has no corresponding language extension~~,~~ ; and

~~wherein~~ converting said second code ~~can be converted~~ into a second source code ~~(SC*)~~ formulated in at least one of the first programming language ~~or~~ and a different programming language and yields at least one of a valid binary code and~~/~~ byte code ~~(B*)~~.

7. (Currently Amended) The method as claimed in claim 6, wherein at least one language extension ~~(LE*)~~ is at least one of newly generated in the second code ~~(CodeML*) or~~ and taken over from the first code ~~(CodeML)~~, and this at least one of generation ~~or~~ and takeover is performed by the language converter ~~(LC)~~.

8. (Currently Amended) A method for modifying source code, comprising:

~~wherein~~ converting a source code ~~(SC)~~, formulated in a first programming language, ~~is converted~~ into a first code ~~(CodeML)~~ formulated in a meta markup language~~,~~ ;

~~wherein~~ transforming the first code, ~~a transformation (T) takes place~~ exclusively in accordance with transformation rules ~~TR~~, into a second code ~~(CodeML*)~~ formulated in the meta markup language; and

~~wherein~~ transforming ~~said~~ the second code ~~is transformed~~ into a second source code ~~(SC*)~~ formulated in at least one of the first programming language ~~or~~ and a different programming language, the first and the second source code differing in terms of their functionality ~~(B, B*)~~.

9. (Currently Amended) The method as claimed in claim 8,

wherein the transformation rules ~~(TR)~~ include at least one condition ~~C~~ and at least one of one logic component ~~L~~ and~~/or~~ code fragment ~~CF~~ itself.

10. (Currently Amended) The method as claimed in claim 8~~ or 9~~, wherein transformation rules ~~(TR)~~ include at least one fragment in the form of at least one of a template ~~(TPF)~~ and~~/or~~ at least one pattern ~~(PF)~~ in which at least one code modification is effected with the aid of the transformation ~~T~~.

11. (Currently Amended) The method as claimed in claim 10, wherein ~~TR is~~is the transformation rules are embodied in such a way that a mechanism for backing up at least one system state is incorporated into the second source code ~~(CodeML*)~~ with the aid of the transformation ~~T~~ in order to enable a migration into other versions.

12. (Currently Amended) The method as claimed in claim 8~~ or 9~~, wherein at least one template ~~(TP)~~ is formed from at least one of the first code ~~or~~ and a fragment of the first code with the aid of the transformation ~~T~~.

13. (Currently Amended) A method for modifying source code, comprising:

~~wherein~~ converting a source code, ~~(SC)~~ formulated in a first programming language, ~~is converted~~ into a first code ~~(CodeML)~~ formulated in a meta markup language~~,~~ ;

~~wherein~~ adding an item of information ~~(INFO)~~ formulated in the meta markup language and influencing the subsequent program execution, ~~(B*) is added by means~~via ~~of~~ a transformation, ~~(T)~~ to ~~said~~ the first code in at least one of a substituting ~~or~~ and non-substituting way and wherein in this way, the second code ~~(CodeML*)~~ also formulated in the meta markup language is formed, the transformation being performed

in accordance with transformation rules ~~(TR)~~ formulated in a transformation description language~~,~~ ; and

~~wherein~~ transforming the ~~said~~ second code ~~is transformed~~ into a second source code ~~(SC*)~~ formulated in at least one of the first programming language ~~or~~ and a different programming language, at least one of the program content and~~/or~~ program execution ~~(B)~~ of the first source code ~~(SC)~~ differing from at least one of the program content and~~/or~~ program execution ~~(B*)~~ of the second source code ~~(SC*)~~.

14. (Currently Amended) The method as claimed in claim 13, wherein said information ~~(INFO)~~ includes at least one code fragment ~~(CFb)~~ and
wherein the second source code is formed in that at least one code fragment ~~(CFa)~~ contained in the first source code is replaced with the aid of the transformation by the at least one code fragment ~~(CFb)~~ contained in the fragment.

15. (Currently Amended) The method as claimed in claim 13, wherein the information ~~(INFO)~~ specifically includes data ~~(D)~~ in the form of at least one of initialization states, ~~(SSDb) or~~ state data ~~(SDa) or~~and database data ~~(Dc)~~.

16. (Currently Amended) The method as claimed in claim 15, wherein the transformation rules ~~(TR)~~ are influenced by ~~said~~ the data ~~(D)~~.

17. (Currently Amended) The method as claimed in ~~one of the~~ claim~~s~~ 13 ~~to 16~~,
wherein at least one of the data ~~(D)~~ and~~/or~~ code fragments ~~(CF)~~ are additionally embedded in the transformation rules.

18. (Currently Amended) The method as claimed in ~~one of the~~ claim~~s~~ 13 ~~to 17~~,

wherein data generated by checkpoints is added by ~~means of~~ a transformation in such a way that the internal state of the original program is at least noe of available at the restart of the program ~~or~~ and is usable~~can be used~~ by ~~said~~ the program.

19. (Currently Amended) The method as claimed in ~~one of the~~ claim~~s~~ 13~~ to 17~~,
wherein information ~~(INFO)~~ includes at least one of updates ~~or~~ and patches.

20. (Currently Amended) The method as claimed in claim 13~~ to 17~~,
wherein fragments ~~(LF1, LF2)~~ contain internationalization information which serves for the adaptation to different natural languages.

21. (Currently Amended) The method as claimed in ~~one of the~~ claim~~s~~ 13~~ to 17~~,
wherein at least one of data and~~/or~~ code fragments originate from a library and
carry information tailored to at least one of customers and~~or~~ customer groups.

22. (Currently Amended) An arrangement for modifying software, comprising:
~~wherein~~ presenting a hybrid form of an original piece of software ~~is present~~ in such a way that at least one part ~~(SC1)~~ of a source text is compiled into at least one of a byte ~~or~~ and binary code ~~(B1,..,B)~~ and at least one further part ~~(SC2)~~ of the source text is converted into a code ~~(CodeML)~~ formulated in a meta markup language for at least one variation point~~ (VP2,..VP),~~;

~~—whereinpresenting~~ a device for transformation ~~(T) is present~~ in such a way that only at least one variation point ~~(VP2)~~ of the hybrid form ~~(SW)~~ of the original software ~~can be converted~~is convertable as necessary ~~by means of~~via the transformation ~~(T)~~ in accordance with transformation rules ~~(TR)~~ into at least one other code ~~(CodeML*)~~ formulated in the meta markup language,
whereby said other code ~~(CodeML*)~~ directly forms a modified variation point ~~(VP2*)~~ of at least one of an adapted piece of software ~~(SW*) or~~and a source code ~~(SC*) can be formed~~is formable from the other code ~~(CodeML*) by means of~~ a converter ~~(RCONV)~~ and then at least one of a binary ~~or~~ and byte code ~~(B*)~~ of the modified variation point ~~(VPB2)~~ of an adapted piece of software ~~(SW*) can be~~is formable ~~formed by means of~~ a compiler ~~(COMP)~~ and whereby the original ~~(SW)~~ and the adapted software ~~(SW*)~~ differ in terms of at least one of their program execution and~~/or~~ program content.

23. (Currently Amended) An arrangement for modifying source code, comprising at least one of:

~~—whereinpresenting~~ a processor ~~is present~~ in such a way that a transformation ~~(T)~~ of the source code ~~(CodeML)~~ is performed in accordance with transformation rules ~~(TR)~~ formulated in an extended style description language and a language converter ~~(LC)~~ contained therein in such a way that either a second code ~~(CodeML*)~~ formulated in the meta markup language without the language extensions ~~(LE)~~ of the first code ~~(CodeML)~~ that were formulated in the meta markup language~~,~~; and
~~or~~ generating a second code ~~(CodeML*)~~ formulated in the meta markup language ~~is generated~~ using at least one of new and~~/or~~ the original language extensions ~~(LE)~~ formulated in the meta markup language, ~~and~~

– wherein a back-converter ~~(RCONV)~~ is present in such a way that ~~said~~ the second code is transformed into a source code ~~(SC*)~~ formulated in at least one of the first programming language ~~or~~ and a different programming language.

24. (Currently Amended) An arrangement for modifying source code, comprising:

~~wherein~~ presenting a first converter ~~(CONV) is present~~ in such a way that a source code ~~(SC)~~ formulated in a first programming language is converted into a first code ~~(CodeML)~~ formulated in a meta markup language~~,~~;

~~wherein~~ presenting a processor ~~is present~~ in such a way that the CodeML is converted ~~by means of~~via a transformation ~~(T)~~ exclusively in accordance with transformation rules ~~(TR)~~ into a second code ~~(CodeML*)~~ formulated in the meta markup language; and

~~wherein~~ presenting a second converter ~~(RCONV) is present~~ in such a way that ~~said~~ the second code is converted into a second source code ~~(CodeML*)~~ formulated in at least noe of the first programming language ~~or~~ and a different programming language, the first and the second source code differing in terms of at least one of their functionality and~~/or~~ content ~~(B, B*)~~.

25. (Currently Amended) An arrangement for modifying source code, comprising:

~~wherein~~ presenting a first converter ~~(CONV) is present~~ in such a way that a source code ~~(SC)~~ formulated in a first programming language is converted into a first code ~~(CodeML)~~ formulated in a meta markup language~~,~~;

~~wherein~~ presenting a processor ~~is present~~ in such a way that an item of information ~~(INFO)~~ formulated in the meta markup language and influencing the subsequent program execution ~~(B*)~~ is added by ~~means of~~ a transformation ~~(T)~~ to ~~said~~ the first

code in at least one of a substituting and~~or~~ non-substituting manner and in this way the second code ~~(CodeML*)~~ also formulated in the meta markup language is formed, the transformation being performed in accordance with transformation rules ~~(TR)~~ formulated in a transformation description language~~,~~ ; and

~~wherein~~ presenting a second converter ~~(RCONV) is present~~ in such a way that ~~said~~ the second code is transformed into a second source code ~~(SC*)~~ formulated in at least noe fo the first programming language ~~or~~ and a different programming language, at least one of the program content and~~/or~~ program execution ~~(B)~~ of the first source code ~~(SC)~~ differing from at least one of the program content and~~/or~~ program execution ~~(B*)~~ of the second source code ~~(SC*)~~.

26. (New) The method as claimed in claim 1, wherein the modification rule initiates an update to at least one of a more recent software version or a patching operation.

27. (New) The method as claimed in claim 1, wherein the modification of at least one variation point is performed by way of the transformation at runtime.

28. (New) The method as claimed in claim 9, wherein transformation rules include at least one fragment in the form of at least one of a template and at least one pattern in which at least one code modification is effected with the aid of the transformation.

29. (New) The method as claimed in claim 8, wherein at least one template is formed from at least one of the first code and a fragment of the first code with the aid of the transformation.